アプリのセキュリティを第一に考える

最新のOWASP TOP 10と その他の脅威に備える

概要

Webアプリケーション ファイアウォールは最近、大きな注目を集めていますが、それには理由があります。大規模なセキュリティおよび情報漏洩をあちこちで何度も耳にしますが、これも同じ理由です。

Webアプリケーション セキュリティは実に困難極まる作業です。

時間やコストがかかるのは言うまでもありません。包括的なWebセキュリティ制御を構築および維持するには、ユーザの効率的な業務に必要な現実のアプリケーション機能を開発するために使える限られた予算の大部分が必要になります。

実際、Webアプリケーション セキュリティは非常に難しく、WhiteHat Securityはその『2017 Application Security Statistics Report』で、平均的なWebアプリケーションには3つの脆弱性があるとしています。[2]侵入検査および問題改善への投資は十分ですか?これらのリスクを理解していますか?これらの脆弱性を軽減する適切なツールを導入していますか?

新しいアプリケーションが出荷されるたびにその改善策を構築および再構築することは難しいので、これらの問題は、いつまでも長期的に遍在し続けます。一般的に、Webアプリケーションの脆弱性を理解および防御するには、セキュリティに特化した専門的知識が必要ですが、これはほとんどの開発者にとって、実際の開発を同時にこなしながら養うことが現実的には不可能なスキルセットです。

幸いなことに、選択肢はあります。適切なツールとサードパーティ制御の整備は、リスクの軽減だけでなく、アプリケーション開発の高速化に大いに役立ちます。

30%

公表されている漏洩事件全体の30%には、WEBアプリケーションへの攻撃が関与しています。[1]

[1] http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/
[2] https://info.whitehatsec.com/rs/675-YBI-674/images/WHS%202017%20Application%20Security%20Report%20FINAL.pdf

OWASP
TOP 10

脆弱性とその対策

WEBアプリケーション セキュリティに関するオープン プロジェクト：
教育による脆弱性軽減の可能性

Webアプリケーション セキュリティの欠点は広まりやすく、見過ごされてはいません。2001年、多くのセキュリティ専門家が団結し、開発者を教育することでWebアプリケーション セキュリティの欠点を軽減することを目指して、Open Web Application Security Project (OWASP) を作りました。OWASPは、Webアプリケーション セキュリティのあらゆる面を扱う方法論、文書、ツールおよびトレーニングを公開する非営利的な国際団体です。

OWASP TOP 10：リスクの分類

OWASPのプロジェクトで最も有名なのが「OWASP Top 10」です。これは、Webアプリケーションに共通する最大のセキュリティ問題を示したリストで、絶えず更新されています。OWASP Top 10の目標は、Webアプリケーションの脆弱性に関するリスクの基本的な分類を提供することです。

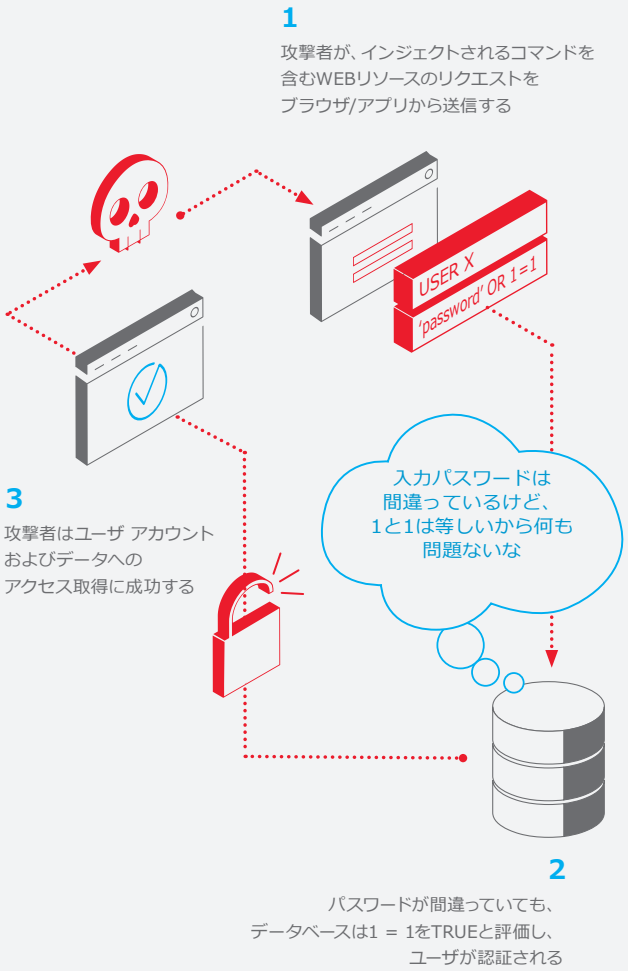
将来的にOWASP Top 10は、その哲学への理解と支持が広まるように、このプロジェクトの信頼性と適応性を向上することを目的として、ISO 31000:2015[3]など、広く受け入れられているリスク フレームワークとより密接に連携していく予定です。

自身のアプリケーションの保護：脅威の概念

Webアプリケーションの開発、セキュリティまたは運用に携わるのであれば、OWASP Top 10への理解を深めることで、そのアプリの保護を強化できます。また、OWASP Top 10に基づくセキュリティ検査は、PCI DSSなど、数多くの業界および規制基準の主要な要件になっています。OWASPサイトでは、OWASP Top 10に関連するその他の国際セキュリティ基準を掲載されています。[4] 最も一般的なWebアプリ セキュリティ問題を掘り下げて研究し、効果的な対策を見つけることで、組織のセキュリティ体制を強化、重要なアプリケーションを保護、およびデータの機密性、整合性、可用性を確保できます。

[3] https://www.iso.org/iso-31000-risk-management.html
[4] https://www.owasp.org/index.php/Industry:Citations

WEBアプリが外部リソースからの入力データを適切にサニタイズしないと、密かにインジェクトされたコマンドの侵入を許してしまいます。



A1 インジェクション

インジェクションは、外部ソースから提供される入力が十分にサニタイズ（文字列をスクリプト コードとして意味を持たないように無害化）されないために、攻撃者からのアプリケーション コマンドが隠される脆弱性の一般的なタイプです。Webアプリケーションはこのような入力を正しくフィルタ処理できないので、インジェクトされたコマンドはローカル システムまたは依存システムのいずれかに侵入できます。

インジェクションの一般的な例はSQLインジェクション攻撃です。多くのアプリケーションは、以下の例に示すように、ユーザからの入力に基づき、SQLステートメントを構築して、情報を取り出したり、ログインを許可したりします。

select * from USERTABLE where USERID = '[userid-from-web-form]' and PASSWORD = '[passwd-from-web-form]'

通常の環境では、これは、データ テーブルUSERTABLEのエントリと一致し、ステートメントが「True」と評価され、ログインが成功します。

しかし、以下のように、ユーザがWebフォームのパスワードに「password' OR 1=1」と入力した場合はどうなるでしょうか。

select * from USERTABLE where USERID = '[userid-from-web-form]' and PASSWORD = 'password' OR 1=1;

適切なサニタイズおよびエスケープが行われない場合、SQLサーバは、1=1の条件をTRUEと評価し、入力されたパスワードに関係なく、入力されたユーザ名でユーザのログインを許可します。これは、非常に簡単で限定的な例です。実際のSQLインジェクション攻撃は、これより遥かに高度で悪意があり、データベース全体を削除、レコードを改竄、および機密データを盗み出すこともできます。

> インジェクションのリスクへの対策

データの入力およびセキュリティに関しては、ユーザからのいかなるデータも本質的には信頼できません。すべての入力データは、検証、エスケープ、サニタイズおよびフィルタ処理が必要です。インジェクション攻撃は、通常のユーザ入力フォームだけでなく、公開されていないWebフィールドでも発生します。パラメータ化されたSQL[5]は、ユーザ入力フォームかどうかに関係なく、入力データがコードに影響を与えないように区別できるので、このリスクを軽減する上で大きな効果を発揮します。また、インジェクション攻撃が成功した場合の情報漏洩を検知できるように、ユーザに返されるアウトバウンド レスポンスを監視することもお勧めします。

[5] https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet#Defense_Option_1:_Prepared_Statements_.28with_Parameterized_Queries.29

A2 認証の不備

ユーザが誰であるか（認証）、および何を許可するか（認可）を正確に認識することは、相互を補完するセキュリティの基本的な概念です。ここでは、Webアプリケーション認証の話なので、パスワードから説明を始めます。パスワードは、仕組みが原始的であり、それに伴うリスクや不便さがあるにも関わらず、残念なことに、いまだにユーザを認証する方法として圧倒的に幅広く利用されています。

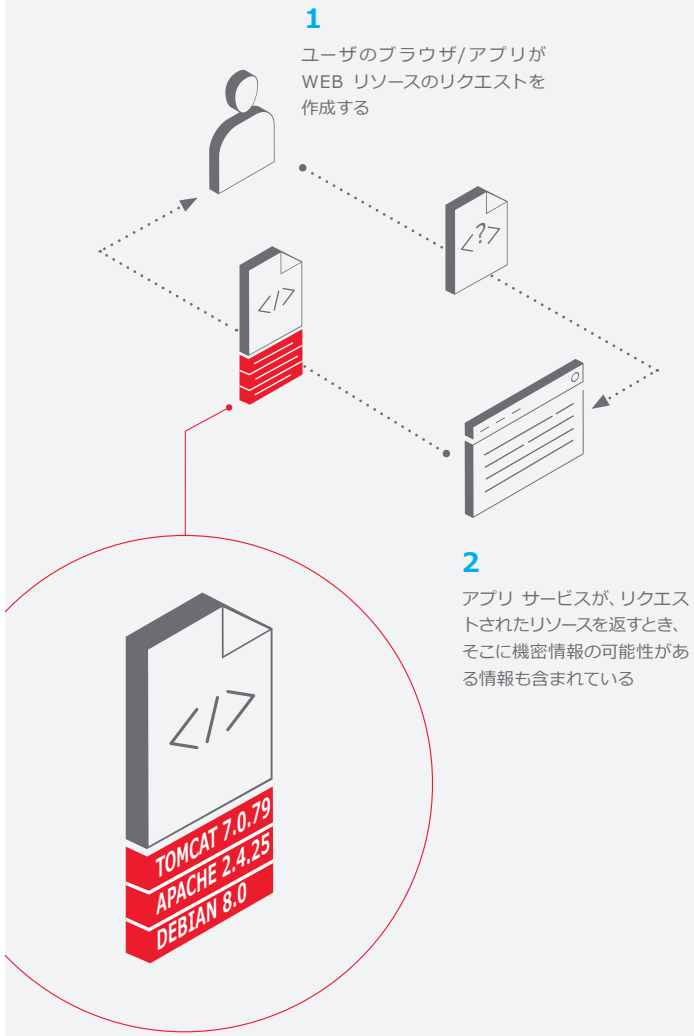
しかし、盗んだ認証情報は、Webアプリケーションの不正アクセスにおいて最も多く利用されている方法です。[7] ユーザ パスワードは、安全性が不十分なため、クレデンシャル スタッフィングなどの攻撃を簡単にするだけなく、その成功に大きく貢献します。クレデンシャル スタッフィング攻撃は、盗まれたユーザ認証情報の大規模なデータベースが攻撃者の手に渡るときに発生します。攻撃者は、自動化ツールを使用して、これらのパスワードをさまざまなサイトやサービスで試し、その効果を確認します。ある推測によると、Fortune 100企業のWebベースのアプリケーションにおける全ログイン試行の90%は、正当なログインでなく、クレデンシャル スタッフィングを目的としたものだとされています。[8] クレデンシャル スタッフィングのような攻撃を可能にしているのは、パスワードから脱すること、またはさらに広く見ればフェデレーション認証を採用することを頑なに拒んでいる私たち自身なのです。[9]

> 認証の不備への対策

パスワードの問題を回避する唯一の方法は、パスワードを使わないことです。クライアント証明書、トークンベースの2要素認証およびフェデレーション認証は、パスワード依存から脱却できる素晴らしい方法です。強力な認証は構築、保護および維持が難しいので、フェデレーション認証を利用することで、アプリケーションの開発および提供を高速化できるだけでなく、ユーザにとってもメリットがあります。どんなにアプリが素晴らしくても、複数のアカウントとパスワードを管理したい人はいません。

[6] https://www.teamsid.com/worst-passwords-2017-full-list/
[7] http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/
[8] http://www.darkreading.com/attacks-breaches/credential-stuffing-attacks-take-enterprise-systems-by-storm/d/d-id/1327908
[9] https://f5.com/about-us/blog/articles/credential-stuffing-what-is-it-and-why-you-should-worry-about-it-24784

WEBサイトは一般的に、必要以上のデータを提供するものなので、悪用できる情報が攻撃者の手に渡っています。



A3 機微な情報の露出

機微な情報の露出は、情報漏洩問題です。漏洩する情報の機密性はさまざまですが、Webアプリケーションの設計に関する情報が攻撃者に漏洩するのは良くありません。このような情報は、自動化されたスキャナで簡単に入手でき、悪用の十分な機会を与えます。

以下に、Webアプリケーションにより漏洩しやすく、攻撃者が役に立つと思う情報の例をいくつか示します。

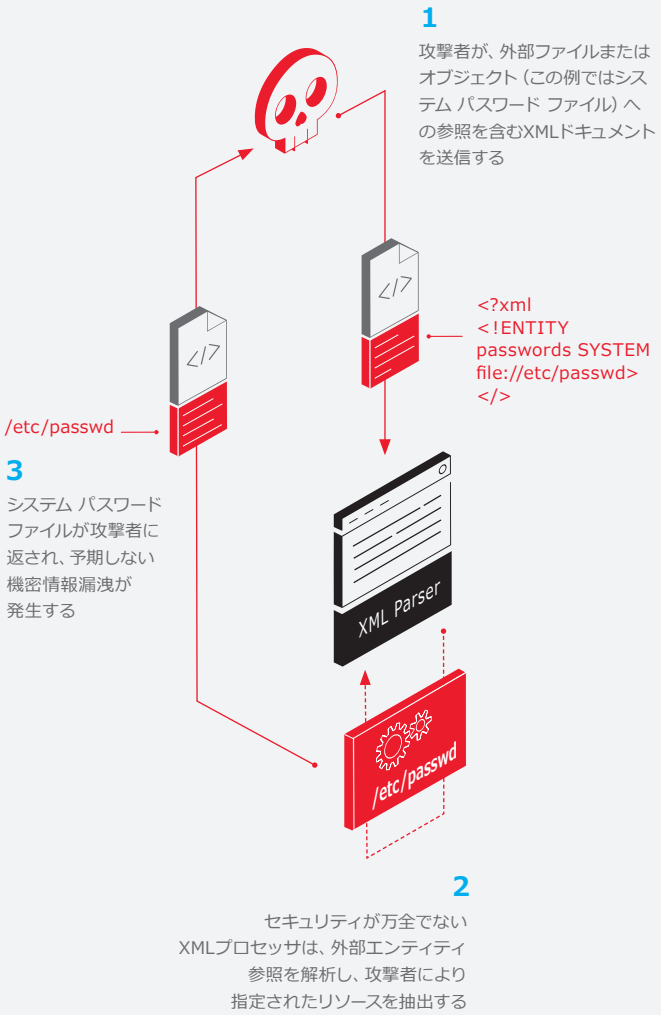
- 予期しない入力がどのように処理されるかを詳しく説明するエラー メッセージ
- サーバ上でのファイルの物理的な位置
- コンポーネントおよびライブラリの具体的なバージョン
- エラーになった関数のスタック トレース（逆コンパイルおよび検証が可能）
- “ユーザIDの妥当性が分かる「Forgot password」関数エラー メッセージ（ユーザ アカウントとパスワードの検出およびブルートフォース（総当たり）攻撃に利用可能）

> 機微な情報の露出への対策

データ漏洩のリスクを軽減できる方法にはいくつかの段階があります。ほとんどのWebサーバはベンダおよびバージョン情報や、その他の情報を報告します。[10] そのため、サーバのレスポンス コードからユーザ名が確認できないようにする必要があります。このためには、不正ユーザ名エラーと不正パスワードエラーで、同じエラー メッセージが生成されるようにします。ブラウザ セキュリティ ディレクティブを使用して、転送中の機密データを保護します。古く、脆弱だと分かっている暗号アルゴリズムおよび暗号方式は使いません。Transport Layer Security（TLS）は、簡単に使用でき、インターネットにおける普遍的な基準になりつつあります。[11] 攻撃者がWebアプリケーションを介してアクセスを取得した場合に備え、暗号化またはトークン化などの技術を使用して、そのWebアプリケーション内に保存されているすべての機密データを読み取りできない状態に処理する必要があります。[12]

[10] https://f5.com/labs/articles/threat-intelligence/identity-threats/phishing-for-information-part-4-beware-of-data-leaking-out-of-your-equipment
[11] https://f5.com/Portals/1/PDF/labs/R065%20-%20REPORT%20-%20The%202016%20TLS%20Telemetry%20Report.pdf
[12] https://www.owasp.org/index.php/File:Reducing_Your_Data_Security_Risk_Through_Tokenization.pptx

XML入力を受け入れるアプリは、XMLプロセッサにデータを漏洩させる外部コマンドを許可してしまうことがあります。



A4 XML外部エンティティ

XMLパーサまたはプロセッサが十分に強化されていない、または正しく設定されていないと、XMLドキュメントの外部エンティティ参照が評価されてしまいます。XML入力を受け取るSOAPサービスなどのアプリでは、予期しない外部参照およびコマンドを指定できることがあり、この場合、XMLプロセッサは、内部ファイル共有のデータを漏洩、コードを実行、内部ポート スキャンを開始、サービス拒否（DoS）攻撃を実行します。

> XML外部エンティティへの対策

まず公開するAPIを慎重に考慮して、脆弱なXMLプロセッサをアップグレード、パッチ処理または他の方法で強化します。次は、HTTPメッセージのサイズ、バージョンおよびメソッドのすべてを強制します。Webアプリケーション ファイアウォール（WAF）を使用することで、適切なリクエストをホワイトリストに追加したり、悪意のあるリクエストがXMLプロセッサに送信されないようにブロックしたりできます。また、WAFにより、JSON、XMLおよびSOAPリクエストの検証が可能になり、DoS攻撃および不正な情報公開を引き起こす既存の攻撃に対してシグニチャを適用できます。さらに、ネットワーク ファイアウォールおよびトラフィック管理により、内部または外部を問わず他のエンドポイントへのアウトバウンド リクエストを制限できます。

A5 アクセス制御の不備

アクセス制御の不備による脆弱性は、機密なオブジェクト（ディレクトリやレコードなど）への不正アクセスに対する強化が正しくまたは十分に行われていないWebアプリケーションの設計上の欠陥です。たとえば、匿名ユーザが、リクエストするURLを知っているだけで、Webサイトの特定のファイルを参照できることもあります。また、実際に確認せずにある程度の認証または権限が発生していることにする機能を、アプリケーションが実行できることもあります。

> アクセス制御の不備への対策

すべてのWebアプリケーション オブジェクトおよびページには、デフォルトでアクセスを拒否する実施メカニズムが必要です。これに基づき、システムは、明示的な権利を試行し、特定のユーザ ロールが割り当てられたユーザにその権利を付与します。

A6 不適切なセキュリティ設定

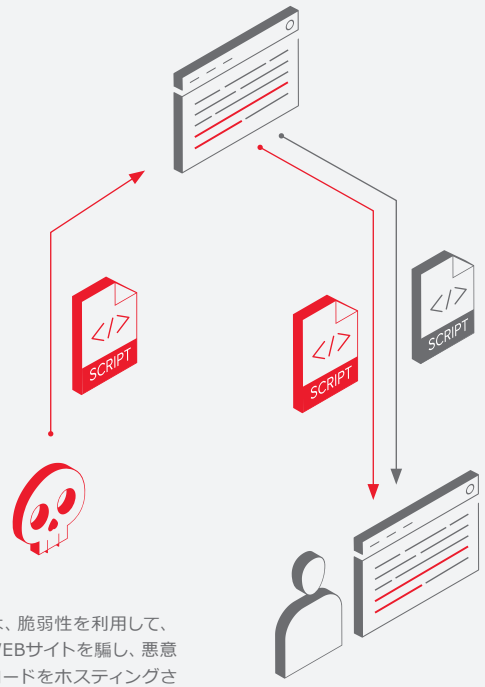
セキュリティ制御の設定が不適切だと考えられる理由はさまざまですが、一般的な原因は、制御に差ができたり、手順が実行されなかったりユーザ側の間違いや不作為です。また、システム管理者による見落としまたは間違いも原因の1つとなり得ますが、結局原因は人間側にあります。依存するソフトウェアおよびインフラストラクチャにも注意が必要です。ほとんどのWebアプリケーションは、他のソフトウェア（Apache、IISまたはNginxなど）に依存し、さらに他のアプリケーション、ライブラリおよびデータベース（PHP、ASPまたはSQL）を利用する場合もあります。

> 不適切なセキュリティ設定への対策

Webアプリケーションを適切に保護するには、ソフトウェア自体だけでなく、他のすべての主要コンポーネントも適切にロックダウンする必要があります。また、制御が正しく実装され、確実に整備されるように、徹底した監査を定期的に実施することも重要です。

[13] http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/

XSSは、外部ユーザがWEBサイトにコンテンツを提供できればどこでも発生でき、脆弱性の最も一般的なタイプの1つになっています。



1

攻撃者は、脆弱性を利用して、正当なWEBサイトを騙し、悪意のあるコードをホスティングさせる

2

ユーザは、信頼できるWEBサイトから、通常のWEBリソースのリクエストを作成する

3

ユーザは、正当なWEBサイトのコードだけでなく、気付かないうちに、攻撃者の悪意のあるコードも受け取り、実行する

A7 クロスサイト スクリプティング（XSS）

クロスサイト スクリプティング（XSS）は、入力検証の脆弱性であり、攻撃者は、訪問するサイトの信頼関係の中で犠牲者のブラウザで独自の悪意あるスクリプトを実行できます。XSSを使用することで、セッション トークンを盗む、隠したトランザクションを開始、または偽造した内容や誤解を招く内容を表示できます。より高度なXSSのスクリプトでは、キーロガーをロードして、犠牲者が入力するパスワードを監視し、その情報を攻撃者が運用する指揮統制サーバに送ることができます。

XSSは、外部ユーザがWebサイトにコンテンツを提供できればどこでも発生でき、脆弱性の最も一般的なタイプの1つになっています。また、XSSは、Webサイトがそのページの表示に必要なHTMLコマンドと同じコマンドを使用するため、識別と排除が困難でもあります。さらに、XSS攻撃はさまざまな方法でエンコードされます。たとえば、ページに「XSS」というメッセージを表示する、以下のような基本的な攻撃スクリプトがあるとします。

<script>alert('XSS')</script>

これをエンコードすると、以下のようになります。

%253Cscript%253Ealert('XSS')%253C%252Fscript%253E

または

<IMG SRC="jav
ascript:alert('XSS');">

さらに以下のようにもなります。

<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A& #x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>

これらすべてのエンコードの可能性はネイティブ ブラウザでサポートされているので、XSSの脆弱性がどのようにしてセキュリティをくぐり抜けるかは一目瞭然です。さらに状況を複雑にしているのが、利用できるXSSの提供方法が多種多様に存在し、攻撃ベクトルを非常に柔軟にしているという事実です。

> クロスサイト スクリプティング（XSS）への対策

XSSを防ぐには、ユーザから提供される外部データは信用できないという考えに戻ります。どのようなデータ入力にも、悪意のあるペイロードが埋め込まれている可能性があるため、すべての着信データをフィルタ処理する必要がありますが、これを各自で包括的に実行することは非常に困難です。しかし幸いにも、優れたWAFは、これを自動的に行うことができるので、再利用可能なインフラストラクチャにより、時間とリソースが解放されます。

A8 安全でないデシリアライゼーション

これはTop 10に新たに追加された項目で、オブジェクトのシリアライゼーションを利用します。シリアライゼーションとは、後で復元できるデータ形式にオブジェクトを変換するプロセスです。ファイルがどのようにディスクに保存されるか、またはデータがネットワーク上でどのように転送されるかを考えます。データは、JSONまたはXMLなどの形式を使用して特定の構造に保存されます。デシリアライゼーションはその逆のプロセスで、この構造化されたデータを読み取り、そこからオブジェクトを構築します。

多くのプログラミング言語は、ネイティブ シリアライゼーションを提供するか、シリアライゼーション プロセスのカスタマイズを可能しますが、これが悪用されています。安全でないデシリアライゼーションは、リモート コード実行、サービス拒否、リプレー、インジェクションおよび権限昇格攻撃を可能にすることがわかっています。

> 安全でないデシリアライゼーションへの対策

デシリアライゼーション攻撃を防ぐには、信頼できないソースからのシリアライゼーションされたオブジェクトを受け入れないことです。また可能であれば、HTTPメッセージ サイズ、バージョン、メソッドおよび必要なヘッダーを強制することもできます。また、高度なWAFは、既知の攻撃シグニチャに対してJSON、XMLおよびSOAPリクエストを検証するだけでなく、データの長さ、値および構造を検証できます。使用するWAFが、機密データとして扱われるデータの処理のカスタマイズを許可する必要もあります。

A9 既知の脆弱性のあるコンポーネントの使用

これも当たり前のように思われるリスク領域ではありますが、多くのソフトウェア コンポーネントが一部の基本的な運用要件を満たす上で有益であるというだけで採用されているため、取り組む価値はあります。このようなコンポーネントの実装時には既知の脆弱性がなかったかもしれませんし、機能性を失う、または価値あるレガシー機能がなくなることを懸念してアップグレードを拒むこともよくあることです。それももっともなことですが、既知のセキュリティ脆弱性が悪用されることで、サービスの重大な損失や顧客の信頼に対する裏切りにつながることを考えれば、このようなリスクは、アップグレードに対する知覚リスクと比較して評価する必要があります。

> 既知の脆弱性のあるコンポーネントの使用への対策

ここで重要なことは、可能であればコンポーネントを最新の状態にしておくことです。これが可能でない場合は、強力なWAFなどの補完制御を利用することで、ソフトウェアを現状のまま運用を続けると同時に、既知の脆弱性の悪用をブロックできます。また、WAFにより、パッチを開発および提供する間の時間を稼ぐこともできます。

A10 不十分なロギングとモニタリング

ほとんどのアプリケーションは、ある程度のアクセスおよび権限情報を記録するように構築されていますが、そうでないアプリケーションも多くあり、デフォルトではそうなっていないアプリケーションもあります。ここでのリスクは、収集または分析されないログ データおよびアクセス情報は、インシデントが発生した場合の漏洩の検知または適切なサービス復旧の促進の助けにならないということです。

攻撃者は、セキュリティへの慢心と盲点を突いて、気付かれないように潜み、アプリおよびネットワークに侵入する足場を確保します。このデータを入念に監視することで、攻撃を早期に検知し、防御姿勢を構築または強化でき、最終的に、組織への損害または影響を最小限に押さえることができます。

> 不十分なロギングとモニタリングへの対策

まず、最も有力なロギング情報を生成し、異常行動の早期警告となるアプリおよびエンドポイントを見つけます。次に、収集する必要がある情報と、それをどのように分析および監視するかを決める必要があります。優れたWAFソリューションを利用することで、Webアプリケーションのすべてのロギングを標準化し、さらなる分析や報告のためにその情報をオフボックスで記録できます。

OWASP TOP 10：
アプリ セキュリティのパズルの1ピースに過ぎない

無数の潜在的な攻撃ベクトルを評価していくと1つのことが明らかになります。それは、Webアプリケーションが対策が困難でコストがかかる数多くの複雑な脅威に直面しているということです。ほとんどの開発チームは、各ベクトルに関連するさまざまな攻撃から十分に防御できるだけのリソースがあるわけではなく、必要な専門的知識のレベルは、プロジェクトに時間と予算があっても入手が困難です。

幸いなことに、高度なWAFテクノロジは、以前よりも入手しやすく、価格も手頃になっています。近代のフル装備のWAFを利用することで、規模を問わずすべての組織は、データセンターでもハイブリッド クラウド環境でも、導入されている重要なアプリを保護できます。一意で柔軟な導入オプションは、実装が簡素化されるだけでなく、アプリに合わせて簡単に保護をカスタマイズできます。

OWASP Top 10を防ぐセキュリティ ニーズに取り組むだけでなく、たとえばDDoS攻撃、ボットによる攻撃、知的財産の盗難および不正行為といった、アプリケーションに対するその他すべての脅威にも対応することが重要です。

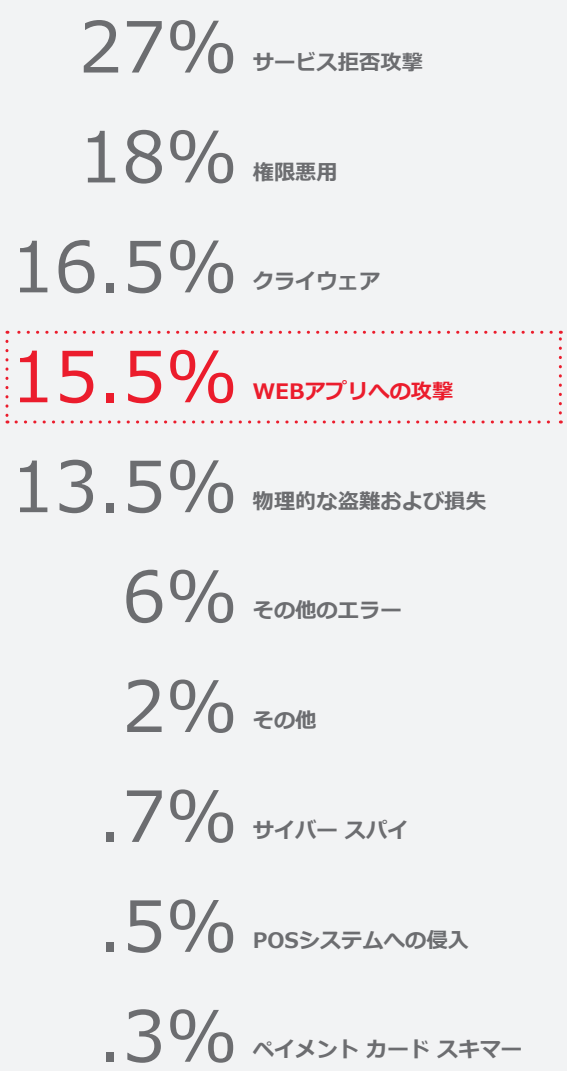
OWASP TOP 10に入らなかった脅威にも注意が必要

たとえば、クロスサイト リクエスト フォージェリ（CSRF）は、ブラウザを使用する犠牲者を騙して一見安全なリンクをクリックさせ、実際には銀行アプリ内で送金を開始するなどの悪意のあるアクションを実行させます。CSRFは、本年度のリストには入っていませんが、犯罪者はCSRFだけでなく、無数の巧妙な手口を利用して、アプリのセキュリティを侵害する機会を狙っています。OWASP Top 10への対策は、素晴らしいことであり、必要なことでもありますが、アプリ、データおよびビジネスの保護に役立つ包括的な多層防御戦略の1つに過ぎません。

アプリケーションや組織に影響を与える脅威、およびそれらの対策の詳細については、f5.com/securityをご覧ください。

サイバー セキュリティ インシデント[14]

パターン別の発生割合および件数



[14] http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/

アプリのセキュリティを第一に考える

常時稼働、常時接続のアプリは、ビジネスを強化および変革する一方、ファイアウォールに保護されないデータへのゲートウェイにもなり得ます。ほとんどの攻撃はアプリ レベルで発生しているため、ビジネスを推進する機能を保護することは、攻撃を受けるアプリを保護することにつながります。

F5ネットワークスジャパン合同会社

東京本社
〒107-0052 東京都港区赤坂4-15-1 赤坂ガーデンシティ19階
TEL 03-5114-3210 FAX 03-5114-3201

西日本支社
〒530-0012 大阪府大阪市北区芝田 1-1-4 阪急ターミナルビル 16 階
TEL 06-7222-3731 FAX 06-7222-3838

お問合わせ先：https://f5.com/jp/fc/

EBOOK-SEC-194147163 | 01.18